Module 2

Relational model and Basic SQL

Syllabus

UNIT 2	Relational Model and Basic SQL	9 hours, P - 6 hours

Relational model: Integrity constraints over relations and enforcement, querying relation data, logical database design, views, destroying/altering tables and views.

Basic SQL: Introduction to SQL, Basic SQL Queries: DML, DDL, DCL, TCL

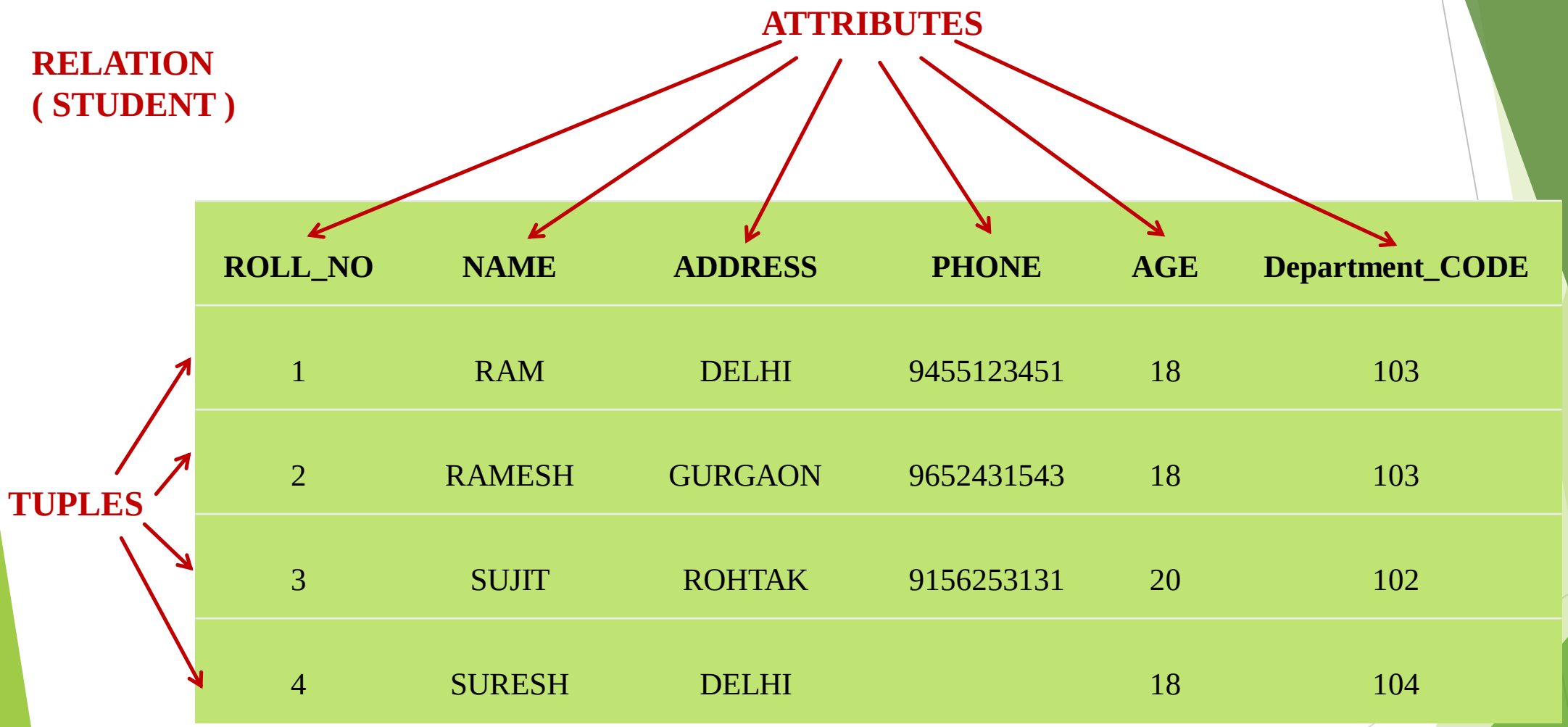
INTRODUCTION

- In relational database management system (RDBMS), data are represented using ***set of rows and columns*** in tabular format.
- Relational model was first introduced by Ted Codd of IBM research in 1970.
- Oracle SQL and MySQL are examples popular RDBMS language to deals with RDBMS data.

INTRODUCTIONS(cont'd)

- The **table** is called a **relation**.
- A row is called a **tuple**.
- A column is called a Field.
- A column header is called an **attribute**.

Relation, Tuple and Attributes



ROLL_NO	NAME	ADDRESS	PHONE	AGE	Department_CODE
1	RAM	DELHI	9455123451	18	103
2	RAMESH	GURGAON	9652431543	18	103
3	SUJIT	ROHTAK	9156253131	20	102
4	SURESH	DELHI		18	104

IMPORTANT TERMINOLOGIES

- **Relation Schema:**
  - Represents **conceptual table** details.
  - Represents name of the relation with its attributes.
  - **Example:** STUDENT (ROLL_NO, NAME, ADDRESS, PHONE, AGE, Department_CODE)
- **Relation Instance:** The **set of tuples of a relation** at a particular instance of time is called as relation instance.
- **Attribute:** Attributes are the properties that define a relation. e.g.; ROLL_NO, NAME

IMPORTANT TERMINOLOGIES

- **Tuple:** Each **row** in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:

1	**RAM**	**DELHI**	**9455123451**	**18**	**103**

- **Degree:** The number of attributes in the relation is known as degree of the relation

- **Cardinality:** The number of tuples in a relation is known as cardinality.

CONSTRAINTS ON RELATION

Constraints

- Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Integrity Constraints**
- An integrity constraint (IC) is a condition that is specified on a database schema, and restricts the data that can be stored in an instance of the database.
- If a database instance satisfies all the integrity constraints specified on the database schema, it is a legal instance.
- A DBMS enforces integrity constraints, in that it permits only legal instances to be stored in the database.

- Integrity constraints are specified and enforced at different times:

1. When the DBA or end user defines a database schema, he or she specifies the ICs that must hold on any instance of this database.
2. When a database application is run, the DBMS checks for violations and disallows changes to the data that violate the specified ICs.

CONSTRAINTS ON RELATION

There are three main integrity constraints −

- Key constraints
- Domain constraints
- Referential integrity constraints

Key Constraints

- There must be at least one **minimal subset of attributes** in the relation, which can **identify a tuple uniquely**.
- This minimal subset of attributes is called key for that relation.
- Key constraints is a relation with a key attribute, **no two tuples can have identical** values for key attributes.
- A key attribute cannot have NULL values.
- Key constraints are also referred to as Entity Constraints.

Candidate Key and Primary Key

- Student schema:

Student ID	First Name	City	Age	Branch	Passport No	Driving License No	DoB

- Minimal set of attributes in a relation that used to differentiate all the tuples of the relations are **candidate keys**.
  - {**Student ID**, Passport No, Driving License No, DoB_FirstName} are candidate keys
- One candidate key can be assigned as **primary key**. Ex: Student ID
- Remaining all other keys are alternative keys.

Key Attribute Set

STUDENT

Stu_id	Name	Branch
11255234	Aman	CSE
11255369	Kapil	ECE
11255237	Aman	CSE
11255678	Aastha	ECE

Domain Constraints

- Attributes have **specific set of values** in real-world scenario.
- For example, age can only be a positive integer.
- Every attribute is bound to have a specific range of values.
- For example, **age cannot be less than zero** and number of **digits in telephone number** must be a fixed number (say 10 digits).

Referential Integrity Constraints

- The **Foreign Key Constraints or** referential integrity constraints is **specified between two relations or tables.**
- Used to maintain consistency among tuples in two relations.
- The concept of **Foreign Keys** introduced by **referential integrity** constraint.
- A foreign key is a key attribute of a relation that can be referred in other relation.

- Referential integrity constraint states that if a relation refers to a key attribute of a different relation, then that key element must exist.

Referential integrity Constraints

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE	Department_CODE
2031	RAM	DELHI	9455123451	18	103
2032	RAMESH	GURGAON	9652431543	18	103
2033	SUJIT	ROHTAK	9156253131	20	102
2034	SURESH	DELHI		18	104

DEPARTMENT

Department_CODE	Department_NAME
103	COMPUTER SCIENCE
105	INFORMATION TECHNOLOGY
102	ELECTRONICS AND COMMUNICATION ENGINEERING
104	CIVIL ENGINEERING

Referential integrity Constraints

- In the STUDENT relation, ‘ROLL_NO’ is the primary key and **‘Department_CODE’ is the foreign key**.
- The foreign key ‘Department_CODE’ referencing another relation DEPARTMENT.
- STUDENT relation is called **referencing relation** in this example.
- DEPARTMENT relation is called **referenced relation** in this table.

Referential integrity Constraints

- **The rules are:**
  - We can't delete a record from a primary table if matching records exist in a related table.
  - We can't enter a value in the foreign key field of the related table that doesn't exist in the primary key of the primary table.
  - We can have a Null value in the foreign key. Because, data may not be available at the time the data is entered.

SPECIFYING CONSTRAINTS IN SQL

Types of constraints available in SQL

- NOT NULL: Enforces a column to **NOT accept NULL values**.
- UNIQUE: Ensures that all values in a column are different.
- DEFAULT: Is used to **set a default value** for a column.
- CHECK: Limit the value range that can be placed in a column.
- Key Constraints – PRIMARY KEY, FOREIGN KEY

SPECIFYING CONSTRAINTS IN SQL

Syntax

```
CREATE TABLE table_name
(
Column_name 1 data type size,
Column_name 2 data type size,
Column_name 3 data type size,
……..
Column_name n data type size
);
```

Example

```
CREATE TABLE STUDENT
(
ROLL_NO INT,
STU_NAME VARCHAR2(35),
STU_AGE INT,
STU_ADDRESS VARCHAR(235)
);
```

or

```
CREATE TABLE STUDENT( ROLL_NO INT, STU_NAME VARCHAR (35), STU_AGE INT, STU_ADDRESS VARCHAR (235));
```

SPECIFYING CONSTRAINTS IN SQL

NOTNULL

```
CREATE TABLE STUDENT
(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (235)
);
```

STUDENT

ROLL_NO	STU_NAME	STU_AGE	STU_ADDRESS
11255369	Aman	CSE	Kerala
11255369	Kapil	ECE	Karnataka
11255237	Aman	CSE	Andra
11255678	Aastha	ECE	NULL

SPECIFYING CONSTRAINTS IN SQL

UNIQUE

```
CREATE TABLE STUDENT
(
ROLL_NO INT UNIQUE,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT,
STU_ADDRESS VARCHAR (35)
);
```

STUDENT

ROLL_NO	STU_NAME	STU_AGE	STU_ADDRESS
11255234	Aman	10	Kerala
11255369	Kapil	20	Karnataka
11255237	Anu	56	Andra
11255678	Aastha	23	Tamilnadu

SPECIFYING CONSTRAINTS IN SQL

DEFAULT

```
CREATE TABLE STUDENT
(
ROLL_NO   INT  UNIQUE,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT ,
EXAM_FEE INT  DEFAULT10000,
STU_ADDRESS VARCHAR (35)
);
```

STUDENT

ROLL_NO	STU_NAME	STU_AGE	EXAM_FEE	STU_ADDRESS
11255234	Aman	CSE	10000	Kerala
11255369	Kapil	ECE	20000	Karnataka
11255237	Aman	CSE	10000	Andra
11255678	Aastha	ECE	30000	Tamilnadu

SPECIFYING CONSTRAINTS IN SQL

CHECK

```
CREATE TABLE STUDENT
(
ROLL_NO   INT  UNIQUE CHECK(ROLL_NO >10000000),
STU_NAME VARCHAR (35)  NOT NULL,
STU_AGE INT  NOT NULL,
EXAM_FEE INT DEFAULT 10000,
STU_ADDRESS VARCHAR (35)
);
```

STUDENT

ROLL_NO	STU_NAME	STU_AGE	EXAM_FEE	STU_ADDRESS
11255234	Aman	CSE	10000	Kerala
11255369	Kapil	ECE	20000	Karnataka
11255237	Aman	CSE	10000	Andra
11255678	Aastha	ECE	30000	Tamilnadu

SPECIFYING CONSTRAINTS IN SQL

KEY CONSTRAINTS

PRIMARY KEY

```
CREATE TABLE STUDENT
(
ROLL_NO   INT ,
STU_NAME VARCHAR (35)  NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY(ROLL_NO)
);
```

STUDENT

ROLL_NO	STU_NAME	STU_AGE	STU_ADDRESS
11255234	Aman	CSE	Kerala
11255369	Kapil	ECE	Karnataka
11255237	Aman	CSE	Andra
11255678	Aastha	ECE	Tamilnadu

SPECIFYING CONSTRAINTS IN SQL

FOREIGN KEY

```
CREATE TABLE STUDENT
(
ROLL_NO   INT ,
STU_NAME VARCHAR (35)  NOT NULL,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35),
PRIMARY KEY(ROLL_NO)
);

CREATE TABLE ENROLLMENT
(
ENROLL_ID INT PRIMARY KEY,
STU_ID   INT,
COURSE_NAME VARCHAR (35),
FOREIGN KEY(STU_ID)REFERENCES STUDENT(ROLL_NO)
);
```

SPECIFYING CONSTRAINTS IN SQL

FOREIGN KEY

STUDENT

ROLL_NO	STU_NAME	STU_AGE	STU_ADDRESS
11255234	Aman	CSE	Kerala
11255369	Kapil	ECE	Karnataka
11255237	Aman	CSE	Andra
11255678	Aastha	ECE	Tamilnadu

ENROLLMENT

ENROLL_ID	STU_ID	COURSE_NAME
101	11255234	DBMS
102	11255234	ToC
103	11255237	OS
104	11255678	DAA

ENFORCING INTEGRITY CONSTRAINTS ON RELATION

- Integrity constraints are specified when a relation is created and enforced when a relation is modified.
- Insert or update command causes a violation of PRIMARY KEY, UNIQUE and Domain constraints.
- Delete commands causes a violation of referential integrity constraints.
- Deletion does not cause a violation of domain, primary key or unique constraints.

CREATE, INSERT, UPDATE, DELETE

STUDENT

sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.2
53666	Jones	jones@cs	18	3.3
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.7
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

UPDATE COMMAND & DELETE COMMAND

- Syntax: **UPDATE *table_name***
  SET *column1* = *value1*, *column2* = *value2*, ...
  WHERE *condition*;
- Syntax: **DELETE FROM *table_name* WHERE *condition*;**

Example: update STUDENT set sid=140 where age=11;

Example: delete from STUDENT where gpa=3.2;

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

ENFORCING INTEGRITY CONSTRAINTS ON RELATION

- **Example 1:**

  INSERT INTO Students (ROLL_NO , STU_NAME , STU_AGE, STU_ADDRESS) VALUES (11255234, ‘Mike’, ‘ECE’, 17, ‘Andra’) ;

- **Example 2:**

  INSERT INTO Students (ROLL_NO , STU_NAME , STU_AGE, STU_ADDRESS) VALUES (NULL, ‘Mike’, ‘ECE’, 17, ‘Andra’) ;

ROLL_NO	STU_NAME	STU_AGE	STU_ADDRESS
11255234	Aman	17	Kerala
11255369	Kapil	18	Karnataka
11255237	Anil	19	Andra
11255678	Aastha	20	Tamilnadu

ENFORCING INTEGRITY CONSTRAINTS ON RELATION

- **Example 3:**

  INSERT INTO Students (ROLL_NO , STU_NAME , STU_AGE, STU_ADDRESS) VALUES (11255240, ‘Mike’, ‘20’, ‘ECE, ‘Andhra’) ;

- **Example 4:**

  UPDATE Students

  SET sid = 11255369

  WHERE sid = 11255237;

ROLL_NO	STU_NAME	STU_AGE	STU_ADDRESS
11255234	Aman	17	Kerala
11255369	Kapil	18	Karnataka
11255237	Aman	19	Andra
11255678	Aastha	20	Tamilnadu

ENFORCING INTEGRITY CONSTRAINTS ON RELATION

- **Example 5:**

  INSERT INTO Enrolled (cid, grade, sid) VALUES ('Hindi101', 'B', 51111);

- **Example 6:**

  DELETE FROM Student

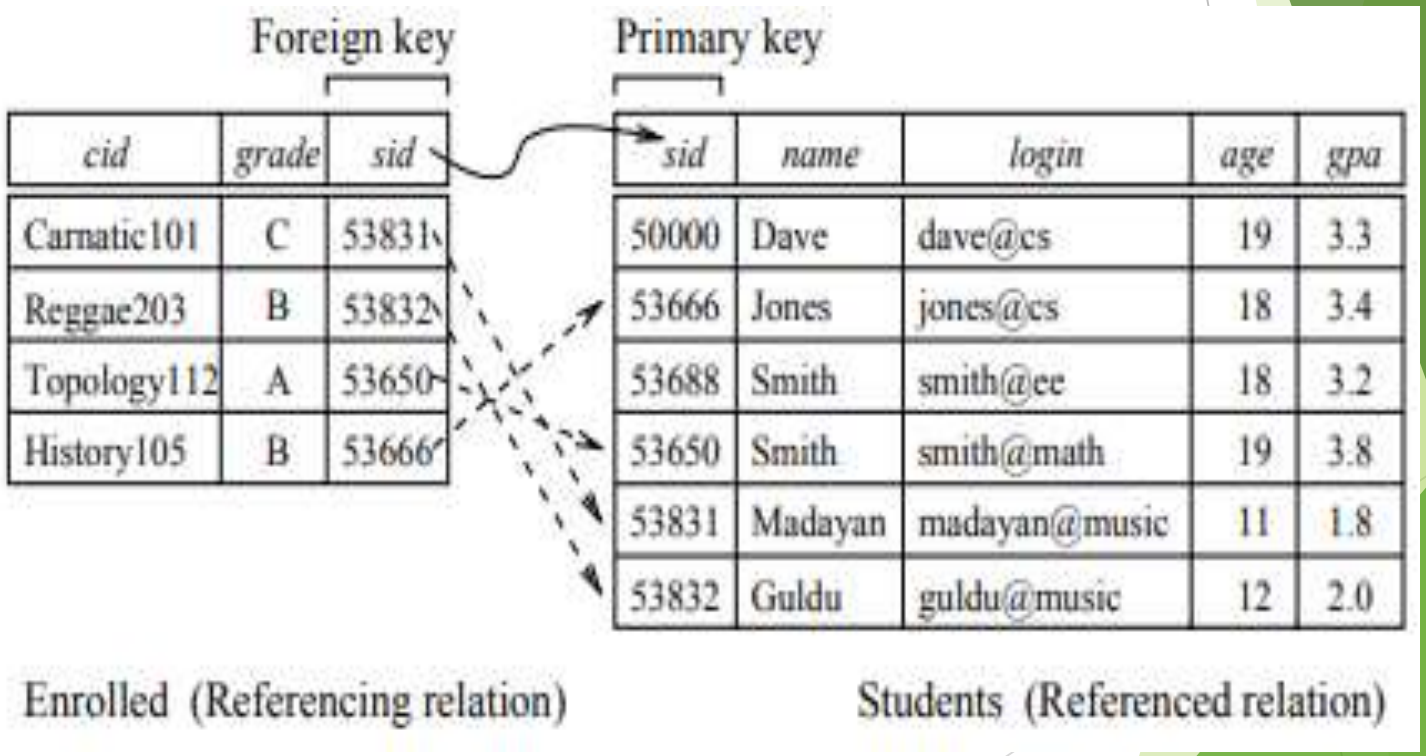
  where sid=53666;

- **Example 7:**

  UPDATE Student

  SET sid= 123

  where sid= 53650;



ENFORCING INTEGRITY CONSTRAINTS ON RELATION

SQL provides several alternative ways to handle foreign key violations.

- **CASCADE** - Delete all Enrolled rows that refer to the deleted Students row. It specifies that the child data is deleted when the parent data is deleted.
- **NO ACTION**- "ON DELETE NO ACTION" Disallow the deletion of the Students row if an Enrolled row refers to it.
- **SET DEFAULT**- Set the sid column to the sid of some (existing) 'default' student, for every Enrolled row that refers to the deleted Students row.
- **SET NULL** - For every Enrolled row that refers to it, set the sid column to null.

ENFORCING INTEGRITY CONSTRAINTS ON RELATION

SQL provides several alternative ways to handle foreign key violations.

```
CREATE TABLE Enrolled
(
sid INT,
cid VARCHAR(20),
grade VARCHAR(10),
PRIMARY KEY (cid),
FOREIGN KEY (sid) REFERENCES Students (sid) ON DELETE CASCADE );
```

ENFORCING INTEGRITY CONSTRAINTS ON RELATION

SQL provides several alternative ways to handle foreign key violations.

```
CREATE TABLE Enrolled
(
sid INT,
cid VARCHAR(20),
grade VARCHAR(10),
PRIMARY KEY (cid),
FOREIGN KEY (sid) REFERENCES Students (sid) ON UPDATE NO ACTION);
```

ENFORCING INTEGRITY CONSTRAINTS ON RELATION

SQL provides several alternative ways to handle foreign key violations.

```
CREATE TABLE Enrolled
(
sid INT DEFAULT 5000,
cid VARCHAR(20),
grade VARCHAR(10),
PRIMARY KEY (cid),
FOREIGN KEY (sid) REFERENCES Students (sid) ON DELETE SET DEFAULT);
```

ENFORCING INTEGRITY CONSTRAINTS ON RELATION

SQL provides several alternative ways to handle foreign key violations.

```
CREATE TABLE Enrolled
(
sid INT ,
cid VARCHAR(20),
grade VARCHAR(10),
PRIMARY KEY (cid),
FOREIGN KEY (sid) REFERENCES Students (sid) ON DELETE SET NULL);
```

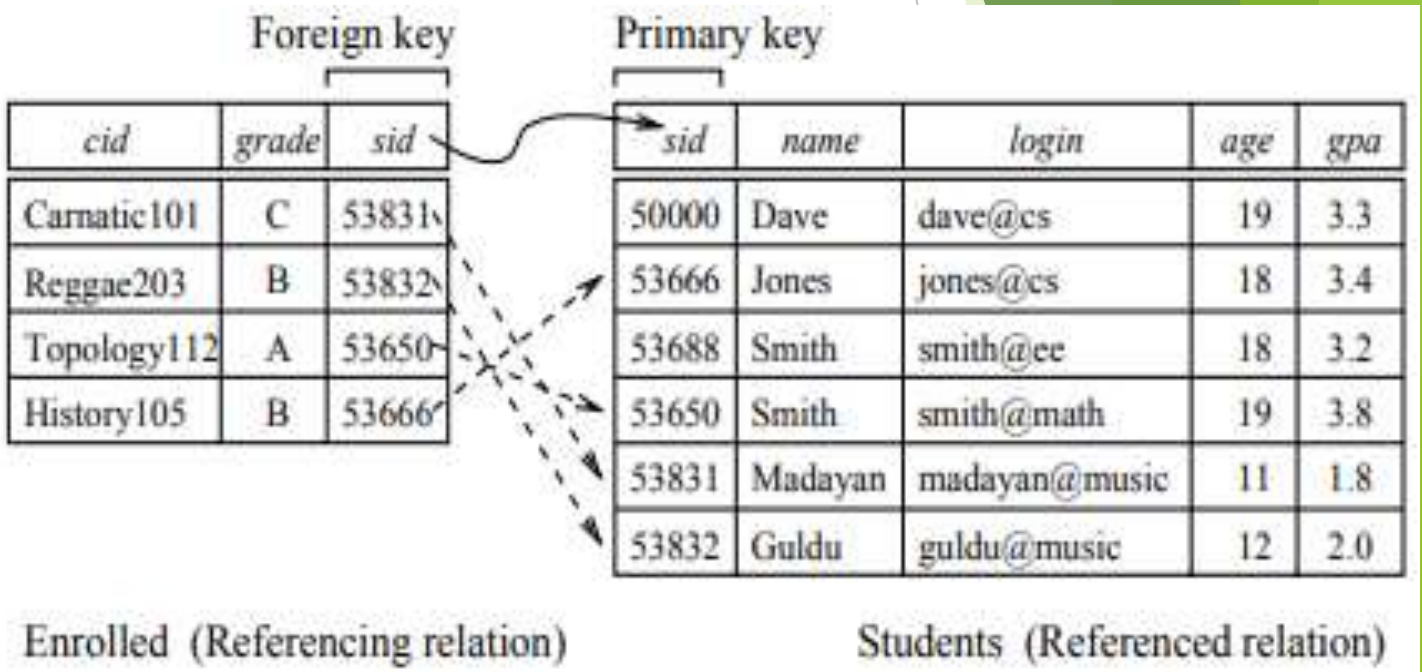
QUERYING RELATIONAL DATA

- A relational database query is a question about the data, and the answer consists of a new relation containing the result.
- A query language is a specialized language for writing queries.
- SQL is the most popular commercial query language for a relational DBMS.

QUERYING RELATIONAL DATA

- **Example 1:** find all students younger than 18

```
SELECT *
FROM Students S
WHERE S.age < 18;
```



Foreign key

Primary key

cid	*grade*	*sid*
Carnatic101	C	53831
Reggae203	B	53832
Topology112	A	53650
History105	B	53666

sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Enrolled (Referencing relation)

Students (Referenced relation)

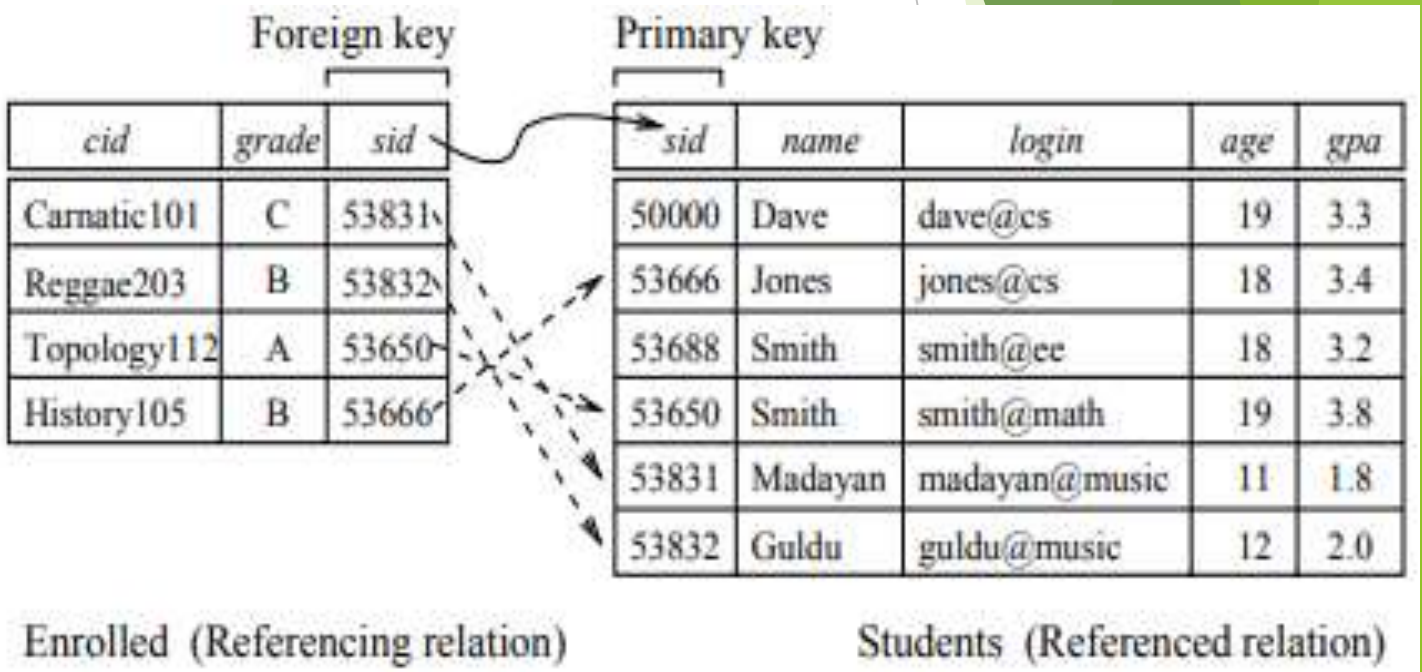
OUTPUT

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

QUERYING RELATIONAL DATA

- **Example 2:** Compute the names and logins of students who are younger than 18.

```
SELECT S.name, S.login
FROM Students S
WHERE S.age < 18;
```



Foreign key

Primary key

cid	*grade*	*sid*
Carnatic101	C	53831
Reggae203	B	53832
Topology112	A	53650
History105	B	53666

sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Enrolled (Referencing relation)

Students (Referenced relation)

OUTPUT

name	*login*
Madayan	madayan@music
Guldu	guldu@music

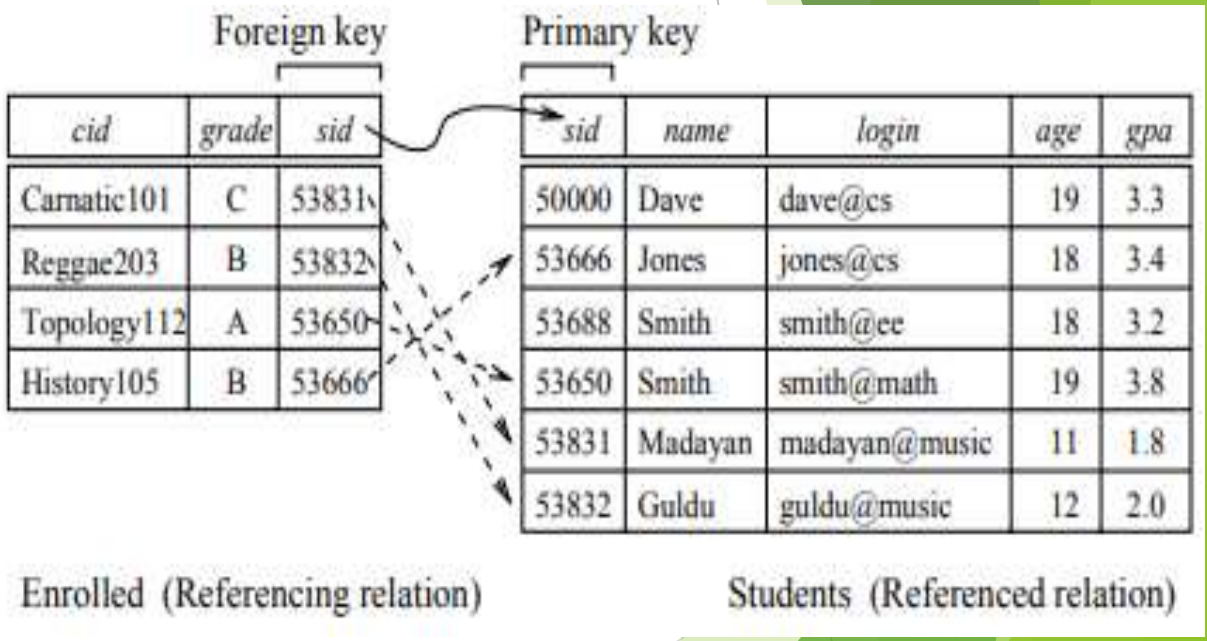
QUERYING RELATIONAL DATA

- **Example 3:** Obtain the names of all students who obtained an grade A and the id of the course in which they got an A.

SELECT S.name, E.cid

FROM Students S, Enrolled E

WHERE S.sid = E.sid AND E.grade = 'A';



OUTPUT

name	*cid*
Smith	Topology112

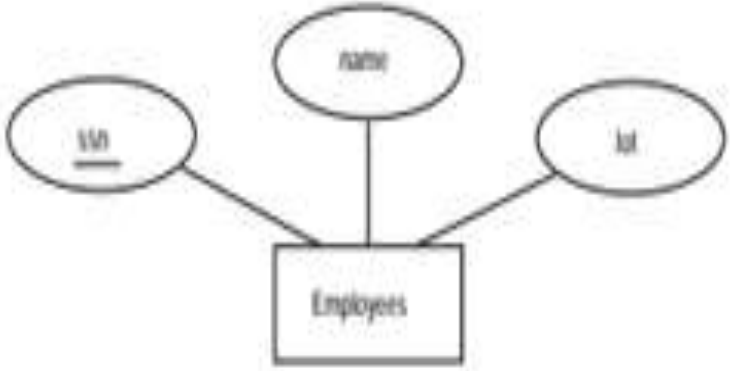
LOGICAL DATABASE DESIGN: ER TO RELATIONAL

- The ER model is convenient for representing an initial, high-level database design.

- Given an ER diagram describing a database, there is a standard approach to generating a relational database schema that closely approximates the ER design.

LOGICAL DATABASE DESIGN: ER TO RELATIONAL

Entity Sets to Tables



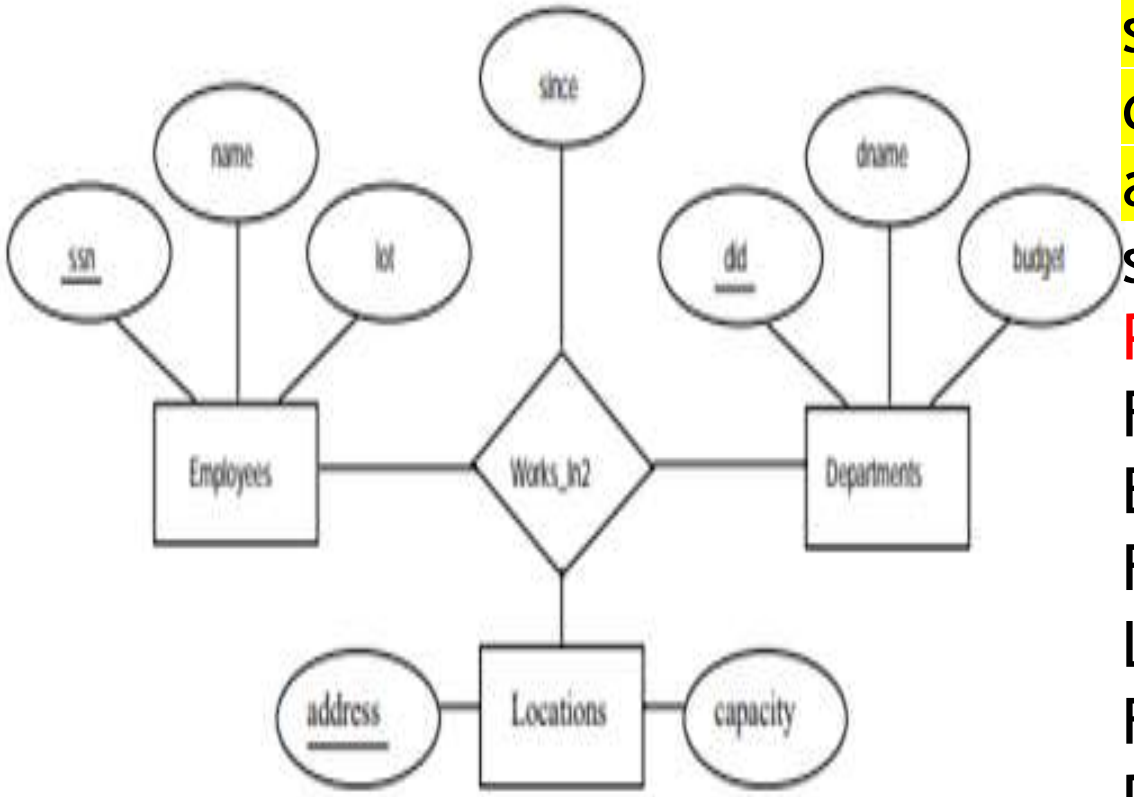
ssn	*name*	*lot*
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

```
CREATE TABLE Employees
(
ssn INT,
name VARCHAR(30),
lot INT,
PRIMARY KEY (ssn)
);
```

Each attribute of the entity set becomes an attribute of the table

ER MODEL TO RELATIONAL DATABASE

Relationship Sets (without Constraints) to Tables




```
CREATE TABLE Works In2
(
ssn INT,
did INT,
address CHAR(20),
since DATE,
PRIMARY KEY (ssn, did, address),
FOREIGN KEY (ssn) REFERENCES
Employees(ssn),
FOREIGN KEY (address) REFERENCES
Locations(address),
FOREIGN KEY (did) REFERENCES
Departments(did)
);
```

- To represent a relationship, we must be able to identify each participating entity and give values to the descriptive attributes of the relationship.

Thus, the attributes of the relation include:

• The primary key attributes of each participating entity set, as foreign key fields.

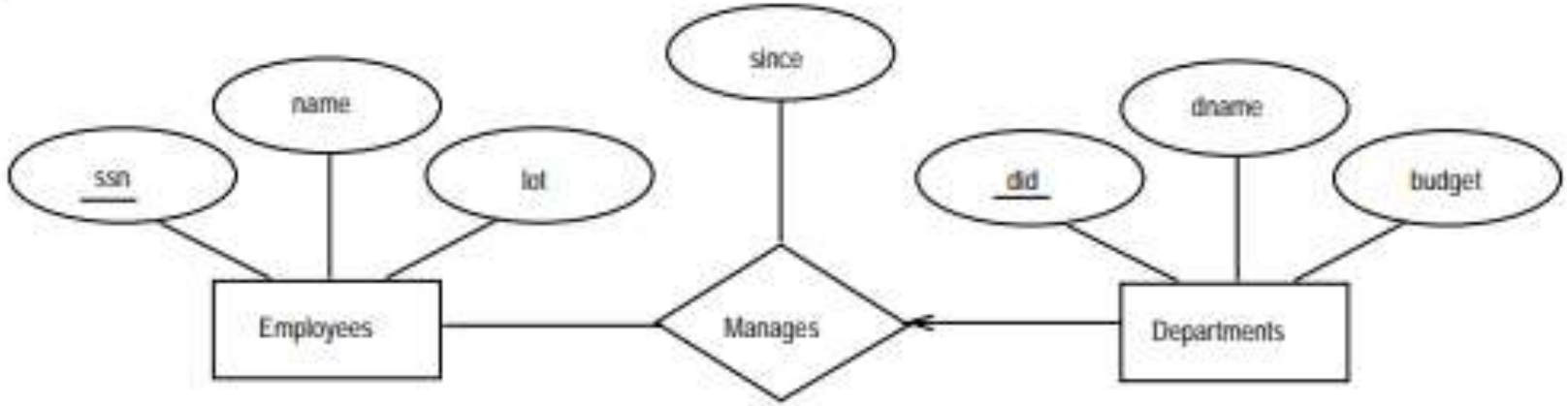
• The descriptive attributes of the relationship set.

- **Ssn,did,address uniquely identify a department**, an employee, and a location in each tuple of WorksIn.

ER MODEL TO RELATIONAL DATABASE

Translating Relationship Sets with Key Constraints

Each department has at most one manager



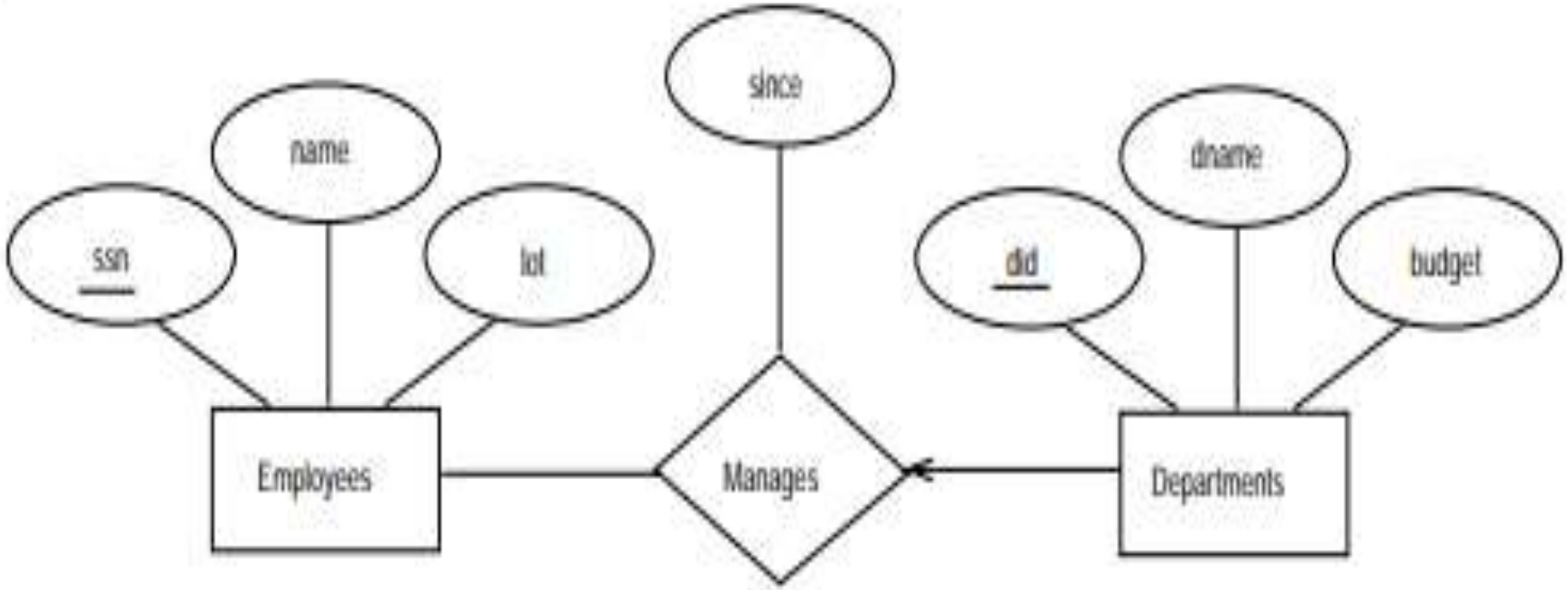

```
CREATE TABLE Manages
(
ssn VACHAR (11),
did INTEGER,
since DATE,
PRIMARY KEY (did),
FOREIGN KEY (ssn) REFERENCES Employees,
FOREIGN KEY (did)REFERENCES Departments);
```

Each department has at most one manager, no two tuples can have the same *did* value but differ on the *ssn* value. A consequence of this observation is that *did* is itself a key for Manages;

ER MODEL TO RELATIONAL DATABASE

Translating Relationship Sets with Key Constraints

Combine manager and Dept in a table. Each department has at most one manager.



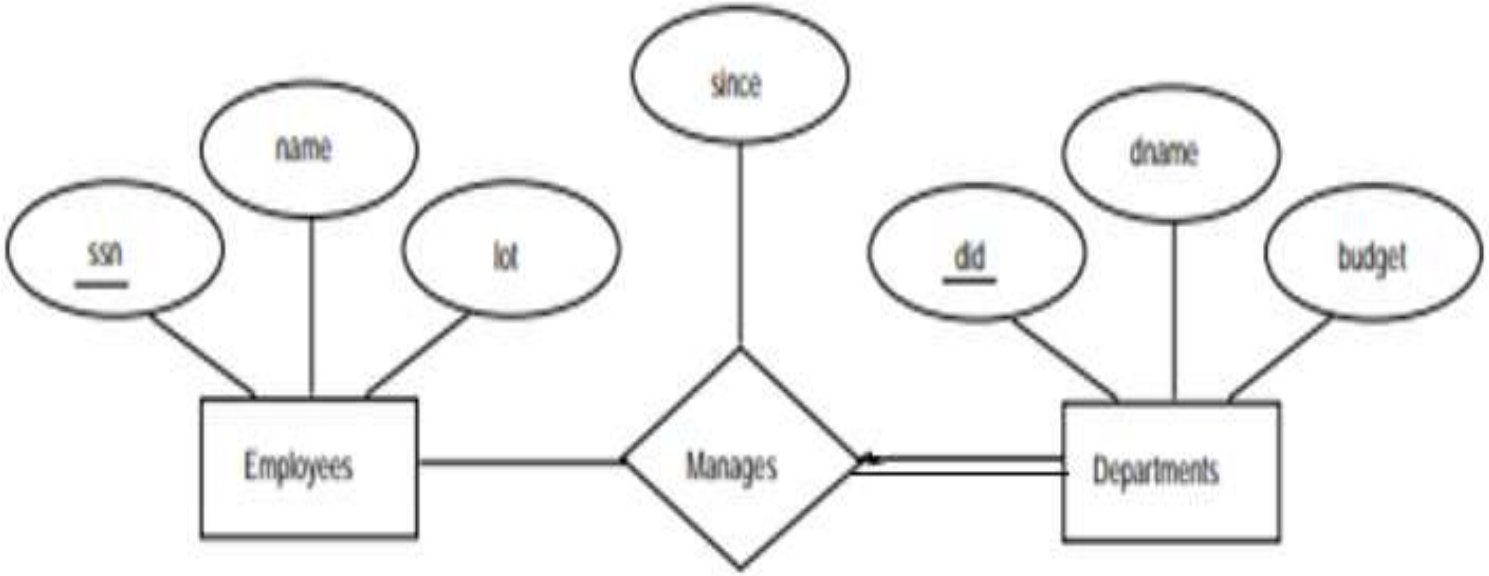

```
CREATE TABLE Dept_Mgr
(
did INT,
dname VARCHAR (20),
budget REAL,
ssn VARCHAR (11) ,
since DATE,
PRIMARY KEY (did),
FOREIGN KEY (ssn) REFERENCES Employees
);
```

- The only drawback to this approach is that space could be wasted if several departments have no managers. In this case the added fields would have to be filled with *null* values.

ER MODEL TO RELATIONAL DATABASE

Translating Relationship Sets with Participation Constraints

- Combine manager and Dept in a table. Every department is required to have a manager.

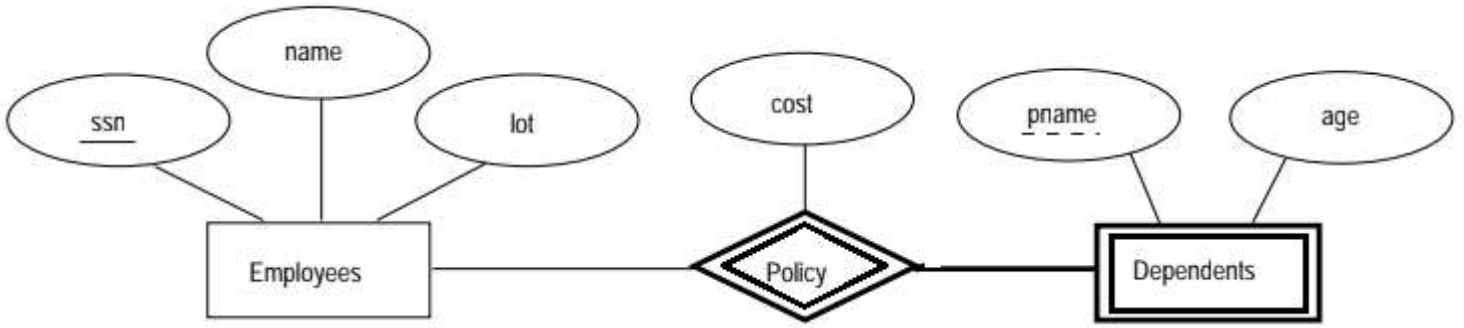



```
CREATE TABLE DeptMgr
(
did INT,
dname VARCHAR(20) ,
budget REAL,
ssn VARCHAR (11) NOT NULL,
since DATE,
PRIMARY KEY (did),
FOREIGN KEY (ssn) REFERENCES Employees ON DELETE NO ACTION
);
```

- It also captures the participation constraint that every department must have a manager: Because *ssn* cannot take on *null* values
- The NO ACTION specification,which is the default and need not be explicitly specified, ensures that an Employees tuple cannot be deleted while it is pointed to by a Dept-Mgr tuple.
- If we wish to delete such an Employees tuple, we must first change the DeptMgr tuple to have a new employee as manager.

ER MODEL TO RELATIONAL DATABASE

Translating Weak Entity Sets. If a tuple in the owning entity is deleted the dependent tuple must be deleted.




```
CREATE TABLE Dep_Policy
(
pname VARCHAR(20) ,
age INT,
cost REAL,
ssn INT,
PRIMARY KEY (pname, ssn),
FOREIGN KEY (ssn) REFERENCES Employees(ssn) ON DELETE CASCADE
);
```

- A weak entity set always participates in a one-to-many binary relationship and has a key constraint and total participation.
- The weak entity has only a partial key. Also, when an owner entity is deleted, we want all owned weak entities to be deleted.
- The CASCADE option ensures that information about an employee's policy and dependents is deleted if the corresponding Employees tuple is deleted.

Translating Class Hierarchy:

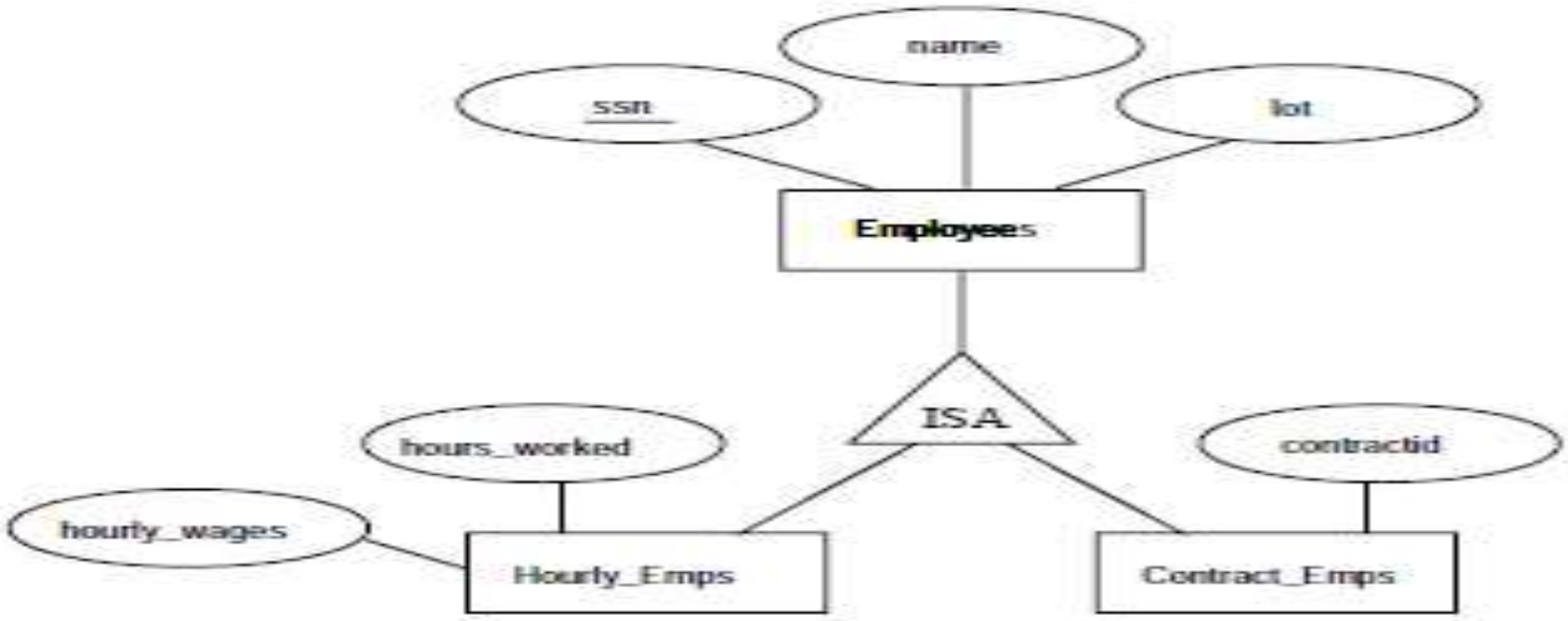


Figure 2.12 Class Hierarchy

- We can map each of the entity sets Employees, Hourly_Emps, and ContractEmps to a distinct relation.

2. The Employees relation is created Hourly_Emps here, ContractEmps is handled similarly. The relation for Hourly_Emps includes the hourly_wages and hours_worked attributes of Hourly_Emps.

3. It also contains the key attributes of the super class (ssn, in this example), which serve as the primary key for Hourly_Emps, as well as a foreign key referencing the super class (Employees).

- For each Hourly_Emps entity, the values of the name and lot attributes are stored in the corresponding row of the super class (Employees). Note that **if the super class tuple is deleted, the delete must be cascaded to Hourly_Emps**.

- 4. Alternatively, we can create just two relations, corresponding to Hourly_Emps and ContractEmps. The relation for Hourly_Emps includes all the attributes of Hourly_Emps as well as all the attributes of Employees (i.e., ssn, name, lot, hourly_wages, hours_worked.
- The second approach is not applicable if we have employees who are neither hourly employees nor contract employees, since there is no way to store such employees.

Translating Aggregation to Relational model:

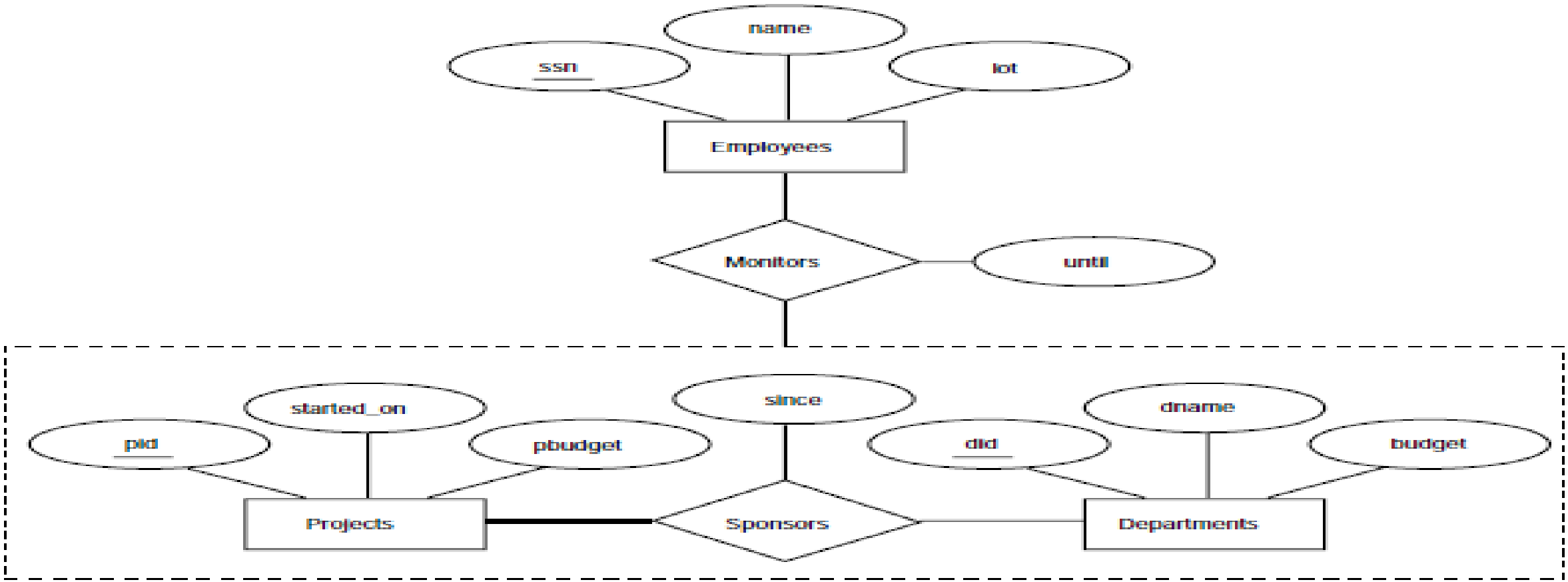


Figure 2.13 Aggregation

- The Employees, Projects, and Departments entity sets and the Sponsors relationship set are mapped.
- For the Monitors relationship set, we create a relation with the following attributes: the key attributes of Employees (ssn), the key attributes (did, pid), and the descriptive attributes of Monitors (until).

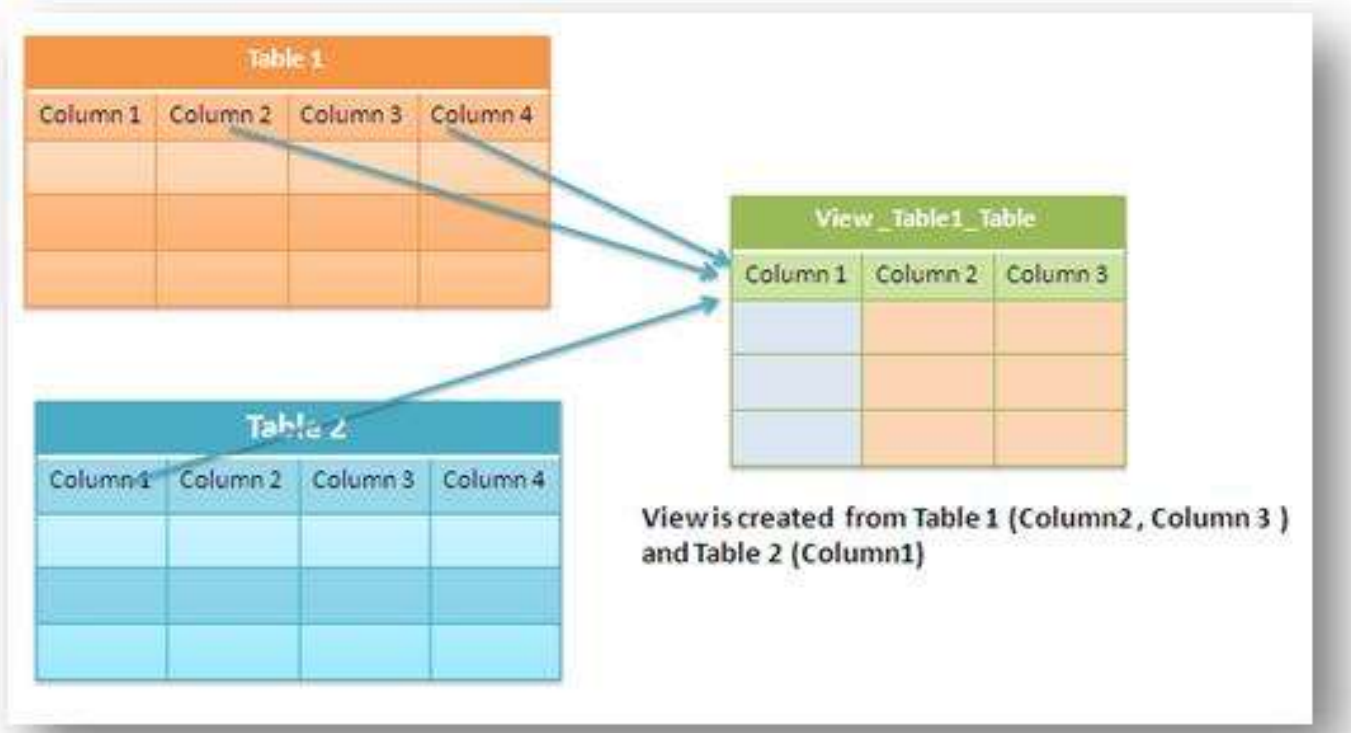
Introduction to Views

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
- Consider a person who needs to know a students name and course enrolled but has no need to see all student details. This person should see a relation described, in SQL, by

CREATE VIEW VIEW_NAME (COLUMN LIST) AS (**select** *sname,cid*
from ***student s***,*enrolled e*
where ***s***.*sid* = *e*.*sid*);

- A **view is a table whose rows are not explicitly stored in the database but are computed as needed.**

- A View is a "Virtual Table". It is not like a simple table, but is a virtual table which contains columns and data from different tables (may be one or more tables)



View Definition

An **SQL View** is a specific representation of data from one or more tables. The tables referred in the views are known as Base tables. Creating a **view** does not take any storage space as only the query is stored in the data dictionary and the actual data is not stored anywhere.

- View is Virtual relation that does not necessarily actually exist in the database but is produced upon request, at time of request.

- The best view for a particular purpose depends on the information the user needs.
- For example, in a telephone directory, a user might want to look up the name associated with a number, without concern for the street address. The best view for this purpose would have two columns: the phone numbers (in numeric sequence) in the first column, and the name associated with each number in the second column.
- Another user might want to look up the phone number associated with a street address, without any need to know the name. The best view for this purpose would have two columns: the street addresses (in alphanumeric order) in the first column, and the phone number in the second column.

Advantages of views

- **Security** Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see.
- **Query Simplicity** A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.
- **Structural simplicity** Views can give a user a "personalized" view of the database structure, presenting the database as a set of virtual tables that make sense for that user.

- **Consistency** A view can present a consistent, unchanged image of the structure of the database, even if the source tables are split, restructured, or renamed.
- **Data Integrity** If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets the specified integrity constraints.

CREATE VIEW B-Students (name, sid, course)

AS SELECT S.sname, S.sid, E.cid

FROM Students S, Enrolled E

WHERE S.sid = E.studid AND E.grade = 'B’

name	*sid*	*course*
Jones	53666	History105
Guldu	53832	Reggae203

- If we try to insert a row (51234, 2.8) into the view, this row can be (padded with *null* values in the other fields of Students and then) added to the underlying Students table.
- We can insert data to a view if the primary key of that table is included in the fields of the view.
- We can update a field of a view if it is obtained from exactly one of the underlying tables, and the primary key of that table is included in the fields of the view.

DESTROYING/ALTERING TABLES AND VIEWS

- If we decide that we no longer need a base table and want to destroy it (i.e., delete all the rows *and* remove the table definition information),

DROP TABLE Students RESTRICT destroys the Students table unless some view or integrity constraint refers to Students; if so, the command fails.

DROP TABLE Students CASCADE

- ALTER TABLE modifies the structure of an existing table. To add a column called *maiden-name* to Students, for example, we would use the following command

ALTER TABLE Students ADD COLUMN maiden-name CHAR(10)

Modifying a Table (Alter Command)

- To change the definition of the table.
- This can be done by using ALTER TABLE command. This command may have one of the following clauses.

- ADD| MODIFY| DROP

Alter Command

- The general syntax for the ALTER TABLE is as follows:

  ALTER TABLE <table name>[ADD|MODIFY| DROP]

  (Constraint | | column specification)

- **ADD Clause:** The ADD clause is used to add a column and/or constraints to an existing table.

- To add a column say part_full_time to emp table, the syntax will be as follows:

  ALTER TABLE emp ADD (part_full_time CHAR(1));

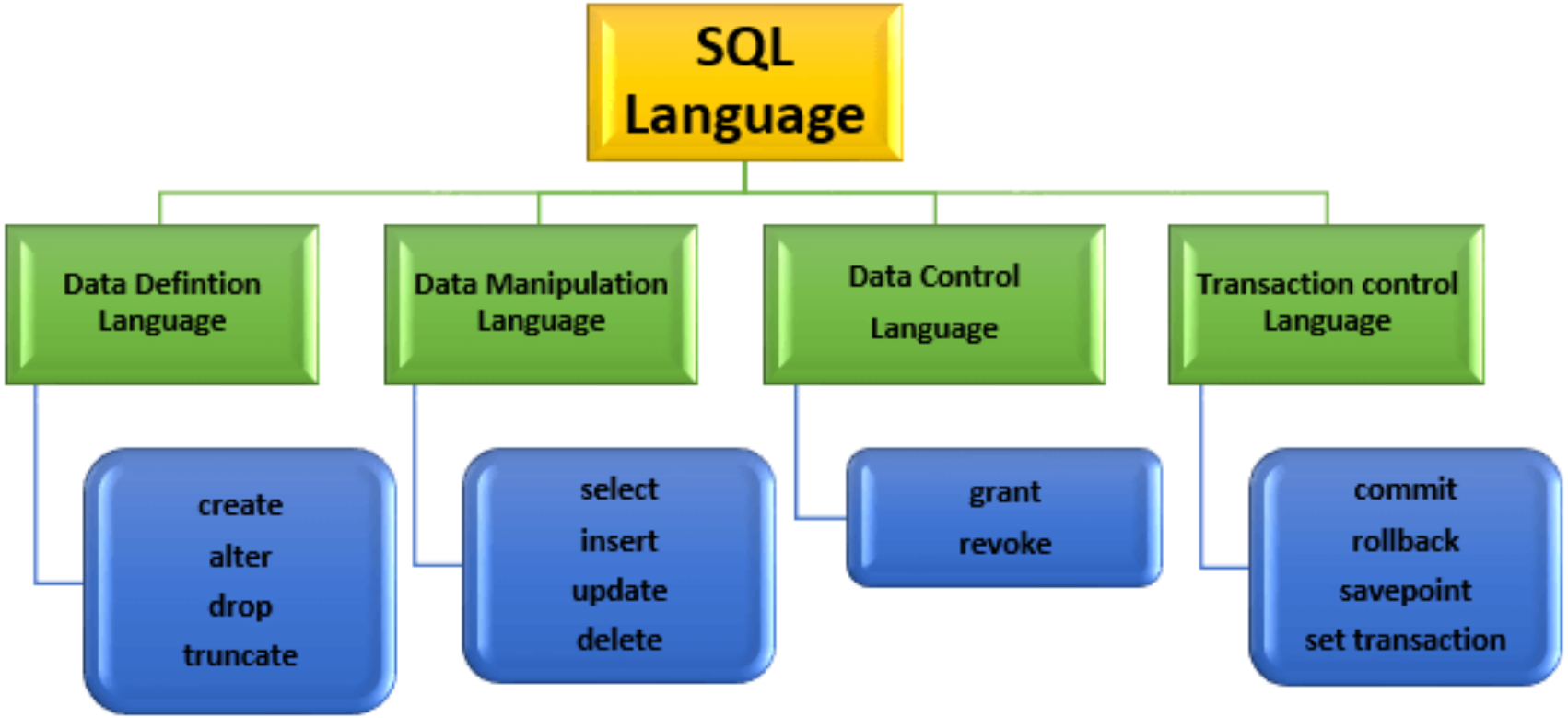
Adding Multiple Columns:

ALTER TABLE emp ADD (part_time CHAR (1), full_time varchar(2));

CHAPTER 2

Basics of SQL

SQL Languages



Structured Query Language(SQL)

- Language for describing database schema and operations on tables
- Structured Query Language(SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database.
- DDL, DML, DCL and TCL are considered sublanguages of SQL

Data Definition Language

- DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema.
- A data definition language (DDL) is a computer language used to create and modify the structure of database objects in a database. These database objects include views, schemas, tables, indexes, etc.

Examples of DDL commands:

- CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- DROP – is used to delete objects from the database.
- ALTER-is used to alter the structure of the database.
- TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.
- RENAME –is used to rename an object existing in the database.

CREATE TABLE

- Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types (INTEGER, FLOAT, DECIMAL(i,j), CHAR(n), VARCHAR(n))
- A constraint NOT NULL may be specified on an attribute

```
CREATE TABLE   DEPARTMENT
 (     DNAME      VARCHAR(10)  NOT NULL,
       DNUMBER INTEGER    NOT NULL,
       MGRSSN     CHAR(9),
       MGRSTARTDATE   CHAR(9)  );
```

Modifying a Table (Alter Command)

- This can be done by using ALTER TABLE command. This command may have one of the following clauses.

- ADD| MODIFY| DROP
- To change the definition of the table.

Alter Command

Alter command is used for altering the table structure, such as,

- To add a column to existing table
- To rename any existing column
- To change data type of any column or to modify its size.
- To drop a column from the table.

- The general syntax for the ALTER TABLE is as follows:

ALTER TABLE <table name>[ADD|MODIFY| DROP]

(Constraint | | column specification)

Alter Command (ADD)

- **ADD Clause:** The ADD clause is used to add a column and/or constraints to an existing table.

- To add a column say part_full_time to emp table, the syntax will be as follows:

  ALTER TABLE emp ADD (part_full_time CHAR(1));

- If this table is having some existing data corresponding to other columns, then it will take NULL for this column corresponding to those records.

Alter Command (ADD)

Adding Multiple Columns:

ALTER TABLE emp ADD (part_time CHAR (1), full_time varchar(2));

MODIFY Clause

- This clause is used to modify the column specifications and the constraints.

- In case of **constraints**, only possibilities are to modify a NULL to NOT NULL and NOT NULL to NULL. Other constraints should be first deleted and then recreated with the modification.

- Suppose we want to **increase** the width of a column, syntax is

  ALTER TABLE emp MODIFY (sal NUMBER(5));

- It will increase the width of sal column from NUMBER(4) to NUMBER(5).

However, for decreasing the width of a column or changing the data type of the column, the column must not contain any data.

MODIFY Clause

- For modification in the constraint, the syntax is:

  ALTER TABLE emp MODIFY(sal NUMBER(5)

  NOT NULL);

- Now, sal column will become NOT NULL column.

DROP Clause

- We can remove a column from table directly by using the DROP clause

  For example:

  ALTER TABLE emp DROP COLUMN part_full_time;

- This will drop the part_full_time column from the table along with the data.

DROP Clause

- To drop a constraint, the different syntaxes are

ALTER TABLE emp DROP CONSTRAINT constraint_name;

// To drop Constraint.

- ALTER TABLE emp DROP PRIMARY KEY;

- DROP TABLE emp; // To drop table along with data permanently.

- ALTER TABLE emp DROP PRIMARY KEY;

//This command will drop all the dependencies of the Primary Key (i.e. the Foreign Key constraints in different table, which are based on this Primary Key) and then will drop this Primary key in a single step.

TRUNCATE Command

- The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table but structure remains.

- Syntax: TRUNCATE TABLE table_name;
- DROP TABLE command delete complete table and remove complete table structure from the database, while TRUNCATE TABLE do not effect table structure .

RENAME

RENAME TABLE:

RENAME TABLE tbl_name TO new_tbl_name;

Where tbl_name is table that exists in the current database, and new_tbl_name is new table name.

RENAME COLUMN: ALTER TABLE tablename RENAME COLUMN OldName TO NewNam

Data Manipulation Language (DML)

- DML is short name of Data Manipulation Language which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE, etc., and it is used to store, modify, retrieve, delete and update data in a database.
- SELECT - retrieve data from a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - Delete all records from a database table

INSERT Command

INSERT Command:- This command is used to insert rows into table.

The basic syntax of this command is as given

INSERT INTO <tablename> (Column1, Column2,…. Column n) VALUES (Value1, Value2…., Value n)

For example ,

INSERT INTO emp(empno, ename, job, sal, hiredate, deptno) VALUES (‘1’,’neha’, ‘student’, ‘34566’, 12-jul-2013,’cse’);

This statement will add a new record in the emp table.

INSERT Command

- When data is not to be entered into every column in the table, then either enter NULL corresponding to all those columns which do not require the value or specify only those columns which require a value.

Example: INSERT INTO emp(empno, ename, job, sal, hiredate, deptno) VALUES ('1','neha', 'NULL', '34566', 12-jul-2013,'NULL');

INSERT INTO emp(empno, ename, sal, hiredate) VALUES ('1','neha', '34566', 12-jul-2013);

INSERT Command

- One more style for INSERT command is without specifying column names as given in the following:

  INSERT INTO emp VALUES (‘1’, ’neha’, ‘student’, ‘34566’, 12-jul-2013,’cse’);

- In this case, values must be in the same sequence as specified at the table creation time.

Changing Table Contents

To change the value of a column or a group of columns in a table corresponding to some search criteria, UPDATE command is used.

- Update can be used for:
  - All the rows from a table
  - A select set of rows from a table.

Update Command

The general syntax of this command is as:

UPDATE<tablename> SET <columnname1> = <newvalue1>, <columnname2> = <newvalue2> WHERE <search criteria>

- For example to update the salary of king to 6000 in emp table, the statement is:

UPDATE emp SET sal = 6000 WHERE ename= ‘king’;

- Example 2:

Update emp set Net_sal = Net_sal + basic_sal*0.15;

Deleting records from the table

- Records from the table can be deleted individually or in groups by using DELETE Command. Delete can be used to:
- Delete all rows from a table.
- A select set of rows from a table.

The general syntax is :

DELETE FROM <tablename> WHERE <search condition>

- For example:

  DELETE FROM emp WHERE deptno=10;

In absence of WHERE Clause, this syntax will delete all the records from the table. **Ex: DELETE FROM emp;**

Delete Command

The subqueries can also be used in DELETE Command for example, if we have to delete all the records from Accounts Department then the syntax will be as follows:

DELETE FROM emp where deptno IN (SELECT deptno from dept WHERE dname= ‘Accounts’);

SQL Select

- To view the global table data.
- To view filtered table data
- Selected column and all rows
- Selected rows and all columns
- Selected columns and selected rows.

To view the global table data

Syntax:

SELECT [DISTINCT] select-list

FROM from-list

WHERE qualification

- The **from-list** in the `FROM` clause is a list of table names. A table name can be followed by a **range variable**; a range variable is particularly useful when the same table name appears more than once in the from-list.

- The **select-list** is a list of (expressions involving) column names of tables named in the from-list. Column names can be prefixed by a range variable.

- The **qualification** in the `WHERE` clause is a boolean combination (i.e., an expression using the logical connectives `AND`, `OR`, and `NOT`) of conditions of the form *expression* `op` *expression*, where `op` is one of the comparison operators $\{<, <=, =, <>, >=, >\}$.[2] An *expression* is a *column* name, a *constant*, or an (arithmetic or string) expression.

- The `DISTINCT` keyword is optional. It indicates that the table computed as an answer to this query should not contain *duplicates*, that is, two copies of the same row. The default is that duplicates are not eliminated.

Description

1. Compute the cross-product of the tables in the from-list.
2. Delete rows in the cross-product that fail the qualification conditions.
3. Delete all columns that do not appear in the select-list.
4. If DISTINCT is specified, eliminate duplicate rows.

Basic

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

Basic Queries

- Find all sailors with a rating above 7.

```
SELECT S.sid, S.sname, S.rating, S.age
FROM   Sailors AS S
WHERE  S.rating > 7
```

Find the names of sailors who have reserved boat number 103

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid = R.sid AND  R.bid=103
```

Find the sids of sailors who have reserved a red boat.

```
SELECT   R.sid
FROM     Boats B, Reserves R
WHERE    B.bid = R.bid AND B.color = 'red'
```

Find the colorS of boats reserved by Lubber.

```
SELECT B3.color
FROM   Sailors S, Reserves R, Boats B3
WHERE  S.sid = R.sid AND R.bid = B.bid AND S.sname = 'Lubber'
```

Find the names of sailors who have Reserved at least one boat.

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid = R.sid
```

LIKE Operator:

- SQL provides support for pattern matching through the LIKE operator, along with the use of the wild-card symbols % (which stands for zero or more arbitrary characters) and _(which stands for exactly one, arbitrary, character).
- E.g. ‘ _AB%’ denotes a pattern that will match every string that contains at least three characters, with the second and third characters being A and B respectively.

LIKE (Examples)

- SELECT * FROM emp WHERE ename LIKE ‘A%’;

  It gives the details of those employees whose name starts from character A. Example: Anu

- SELECT * FROM emp WHERE ename LIKE ‘%a’;

  It returns the rows in which the value in ename column ends with character ‘a’. Example: deepika

- SELECT * FROM emp WHERE ename LIKE ‘%r%’;

  It will display the information about those employees who include ‘r’ in their names.

  Example: priya

Find the ages of sailors whose name begins and ends with B and has atleast three characters.

- SELECT S.age FROM WHERE Sailors S S.sname LIKE 'B_%B’

LIKE allows for a comparison of one string value with another string value

DCL (Data Control Language)

- DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of DCL commands:

- **GRANT:**
- **REVOKE**

GRANT:

- **GRANT:** This command gives users access privileges to the database.
- This is a SQL command which is used to provide privileges/permissions to modify and retrieve database objects like tables, views, sequences, indexes, and synonyms. This command also gives privileges like providing the same permissions to some third user as well.

- **Syntax:**

GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

REVOKE:

- This command withdraws the user's access privileges given by using the GRANT command.
- The REVOKE command in SQL is used to revoke or withdraw permissions that were previously granted to an account on a database object. Therefore, we can think of REVOKE as a tool to limit a role's or user's ability to perform SELECT, INSERT, DELETE, UPDATEand CREATE statements on database objects., as well as to set constraints like foreign keys and update data records, among other operations.
- Syntax:

  REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

TCL (Transaction Control Language)

- In SQL, TCL stands for **Transaction control language**.
- A single unit of work in a database is formed after the consecutive execution of commands is known as a transaction.
- There are certain commands present in SQL known as TCL commands that help the user manage the transactions that take place in a database.
- **COMMIT, ROLLBACK** and **SAVEPOINT** are the most commonly used TCL commands in SQL.

COMMIT

- COMMIT command in SQL is used to save all the transaction-related changes permanently to the disk.
- Whenever DDL commands such as INSERT, UPDATE and DELETE are used, the changes made by these commands are permanent only after closing the current session.
- So before closing the session, one can easily roll back the changes made by the DDL commands.
- Hence, if we want the changes to be saved permanently to the disk without closing the session, we will use the commit command.
- Syntax:

```
COMMIT;
```

SAVEPOINT:

- A SAVEPOINT is a point in a transaction in which you can roll the transaction back to a certain point without rolling back the entire transaction.
- To identify a point in a transaction to which you can later roll back.
- Using the SAVEPOINT command in SQL, we can save these different parts of the same transaction using different names.
- Syntax:

```
SAVEPOINT <savepoint_name>;
```

ROLLBACK:

- One can make use of this command if they wish to undo any changes or alterations since the execution of the last COMMIT
- The ROLLBACK command is used to undo a group of transactions.
- Syntax

```
ROLLBACK TO savepoint_name;
```

THANK YOU